BAB II
LANDASAN TEORI

2.1 Kriptografi

Kriptografi (*cryptography*) berasal dari bahasa Yunani, berasal dari dua suku kata yaitu *kripto* yang artinya menyembunyikan dan *graphia* yang artinya tulisan. Kriptografi adalah ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi, seperti kerahasiaan data, keabsahan data, integritas data, serta autentifikasi data (Munir, 2006).

Menurut Sentot Kromodimoejo, kriptografi adalah ilmu mengenai teknik enkripsi dimana data diacak menggunakan suatu kunci enkripsi menjadi sesuatu yang sulit dibaca oleh seseorang yang tidak memiliki kunci dekripsi (Kromodimoejo, 2010).

2.1.1 Komponen Kriptografi

Cryptography terdiri dari beberapa komponen yaitu :

1. Enkripsi; enkripsi ialah pengamanan atas data yang dikirimkan agar rahasia tetap Aman dengan cara mengacak pesan . *plaintext* atau Pesan aslinya diubah jadi karkater yang tidak dipahami atau dimengerti.
2. Dekripsi; dekripsi ialah mengembalikan data setelah dienkripsi seperti semula (*plaintext*), Algoritma yang digunakan untuk dekripsi tentu tidak sama dengan yang digunakan untuk enkripsi.
3. Kunci; maksud kunci disini ialah kunci yang digunakan untuk mengenkripsi dan dekripsi. Kunci ada dua bagian yaitu kunci pribadi (*private key*) dan kunci umum (*publik key*).
4. Chipertext; ialah suatu pesan yang sudah melalui proses enkripsi. Pesan yang ada pada *chipertext* tidak bisa dibaca karena menjadi karakter-karakter yang tidak memiliki arti (makna).

5. Plaintext; disebut juga cleartext; ialah pesan bermakna yang ditulis dan plaintext ini akan menjadi *chipertext* setelah diproses oleh algoritma *cryptography* tertentu
6. *Cryptanalysis*; maksudnya menganalisa suatu kode atau sandi untuk mendapatkan plainteks tanpa kunci yang sah.

2.1.2 Tujuan Kriptografi

Kriptografi bertujuan memberikan *security* atau keamanan yang memiliki aspek sebagai berikut (Munir, 2006):

1. Kerahasiaan (*confidentiality*), adalah layanan agar pesan hanya dapat dibaca oleh pihak-pihak yang memiliki hak dengan cara merubah pesan menjadi cipherteks.
2. Integritas data (*data integrity*), adalah layanan yang menjamin keutuhan pesan dan tidak ada yang berubah saat pengiriman data.
3. Otentikasi (*authentication*), adalah layanan yang berhubungan dengan pengenalan, baik mengenali kebenaran pihak yang melakukan komunikasi (*user authentication* atau *entity authentication*) maupun mengenali sumber pesan (*data origin authentication*).
4. Nirpenyangkalan (*non-repudiation*), adalah layanan untuk mencegah pengirim pesan melakukan menyangkalan melakukan pengiriman atau penerima pesan melakukan menyangkalan telah mendapat pesan.

2.2 Audio WAV

WAV atau *waveform audio format* merupakan standar format audio yang dikembangkan oleh Microsoft dan IBM. File WAV ini merupakan bagian dari spesifikasi RIFF Microsoft yang digunakan sebagai penyimpanan data digital audio (Kabal, 2006). Format file WAV merupakan salah satu format file audio pada PC.

Format WAV dipilih karena WAV merupakan format utama untuk menyimpan data audio mentah dan pada umumnya belum dikompresi. File Wav biasanya menggunakan *coding* PCM (*Pulse Code Modulation*) sehingga audio analog yang didigitalkan lebih detail dan berukuran lebih besar dari format video

lainnya. Kelebihan dan kekurangan format audio ini sangat mendukung penelitian ini dibandingkan format audio lain.

2.3 Merkle-Hellman Knapsack

Merkle-Hellman adalah algoritma kriptografi dengan kunci asimetris, yang berarti bahwa ada dua kunci yang dibutuhkan untuk berkomunikasi yaitu sebuah kunci publik untuk proses enkripsi dan sebuah kunci private untuk deskripsi. Cara kerja algoritma Merkle-Hellman berdasarkan pada algoritma *knapsack problem* (Merkle & Hellman, 1978).

2.3.1 Algoritma Knapsack

Cara kerja algoritma *knapsack* adalah dengan menempatkan beberapa benda agar dapat dimasukkan ke dalam suatu tempat dengan kapasitas tertentu secara maksimal. Benda tersebut sebanyak *n* (*I1,...,In*) akan di masukkan ke dalam sebuah tempat dengan kapasitas *C*. Jika setiap benda *Ij* memiliki beban *wj* dan keuntungan *bj* maka setiap benda tersebut akan dimasukkan ke dalam tempat dengan tujuan tidak melebihi kapasitas beban karung tersebut dan mendapati keuntungan yang maksimal.

Secara umum, permasalahan *knapsack* dapat dituliskan dalam bentuk:

$$C = I1+I2+I3+\ldots In \text{ , atau } C = b1w1+b2w2+b3w3+\ldots bnwn. \qquad (2.1)$$

Keterangan :
C = Kapasitas
L = Benda
b = Keuntungan
w = Beban

Persoalan merupakan persoalan yang tidak dapat dipecahkan dalam orde waktu polinomial.

Ide dasar algoritma kriptografi *knapsack* adalah dengan menyandikan pesan untuk menjadi rangkaian solusi dari permaslahan *knapsack*. Setiap beban *wi* di dalam persoalan *knapsack* merupakan kunci privat, sedangkan bit-bit plainteks dinyatakan sebagai *bj*.

Namun, algoritma *knapsack* sederhana ini memiliki kekurangan. Pesan yang telah menjadi cipherteks tidak dapat dideskripsikan kembali menjadi pesan awal. Untuk memecahkan permasalahan ini dibentuklah *superincreasing knapsack*.

2.3.2 *Superincreasing Knapsack*

Barisan *superincreasing* adalah sebuah barisan bilangan yang nilai setiap bilangannya lebih besar dari jumlah semua nilai sebelum nya. Sebagai contoh, {2, 3, 6, 13, 29, 60} merupakan barisan *superincreasing* sedangkan {2, 3, 5, 9, 29, 35} bukanlah barisan *superincreasing.*

S*uperincreasing knapsack* dapat ditentukan dengan langkah-langkah sebagai berikut:

a. Jumlahkan semua nilai, beban, atau bobot di dalam barisan tersebut.
b. Bandingkan nilai total dengan nilai terbesar di dalam barisan tersebut. Jika nilai terbesar lebih besar atau sama dengan nilai total, maka nilai tersebut dimasukkan ke dalam *knapsack*. Jika lebih kecil, maka nilai tersebut tidak dapat dimasukkan.
c. Kurangi nilai total dengan beban yang telah dimasukkan. Setelah itu, bandingkan kembali nilai total sekarang dengan nilai terbesar selanjutnya. Langkah tersebut akan terus dilakukan sampai seluruh nilai di dalam barisan selesai dibandingkan.
d. Jika nilai total telah menjadi nol, maka terdapat solusi dari persoalan *superincreasing knapsack.* Akan tetapi, jika nilainya tidak nol, maka tidak terdapat solusi untuk permasalahan tersebut.

Dalam langkah-langkah tersebut, *T* merupakan beban total. Iterasi atau pengulangan akan terus dilakukan sehingga solusi ditemukan, yaitu nilai total *T=0* dan terdapat himpunan solusi *knapsack*.

Superincreasing knapsack pun sebenarnya masih mudah untuk didekripsi sehingga untuk memproteksi pesan keamananya kurang baik. Hal ini disebabkan jika elemen-elemen dari *superincreasing knapsack* diketahui, orang lain akan dapat mengetahui pola bit dari nilai totalnya.

2.3.3 Pengembangan Algoritma *Superincreasing Knapsack*

Algoritma *nonsuperincreasing knapsack* atau *knapsack* normal adalah algoritma yang komputasinya sangat sulit. Karenanya, pemecahannya pun cukup sulit dengan membutuhkan waktu dalam orde eksponensial (*On*).

Martin Hellman dan Ralph Merkle memodifikasi algoritma *superincreasing knapsack* menjadi algoritma *knapsack* normal yang kemudian Algoritma tersebut diberi nama Merkle-Hellman.

Algoritma Merkle-Hellman menggunakan kunci publik dan kunci privat untuk memodifikasi *superincreasing knapsack* menjadi *knapsack* normal. Kunci publik dibentuk dari barisan *non-superincreasing* yang dimanfaatkan untuk enkripsi pesan dan kunci privat dibentuk dari barisan *superincreasing* untuk dekripsi pesan.

Berikut adalah cara kerja algoritma Merkle-Hellman untuk membangkitkan kunci publik dan kunci privat:

a. Tentukan himpunan angka yang merupakan barisan *superincreasing*.
b. Kalikan setiap elemen di dalam barisan tersebut dengan *r* modulo *q*. Modulus *q* harus bernilai lebih besar daripada jumlah semua elemen di dalam barisan *superincreasing* tersebut. Nilai *r* tidak boleh memiliki faktor persekutuan dengan *q*.
c. Hasil perkalian akan menjadi kunci publik sedangkan barisan *superincreasing* semula akan dijadikan kunci privat.

2.3.4 Merkle-Hellman

Algoritma Merkle Hellman mengenskripsi n-bit pesan dengan menggunakan kunci public bilangan prima. Kriptogram atau cipherteks c dirumuskan sebagai:

$$\boldsymbol{c} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{a_i}\,\boldsymbol{\beta_i}. \qquad (\mathbf{2.2})$$

Keterangan :
c = *chipertext*
α = pesan
β = kunci

Pada tahap pertama, penerima pesan membuat dan menentukan kunci publik serta elemen-elemen rahasia lainya. Dalam algoritma Merkle Hellman penerima pesan merupakan pihak yang mengatur keseluruhan sistem. Pihak penerima jugalah yang menentukan barisan *superincreasing* w dengan rumus:

$$\boldsymbol{w_i > \sum_{j=1}^{i=1} w_j.} \qquad \boldsymbol{(2.3)}$$

w mewakili *superincreasing knapsack* yang dapat diselesaikan dalam waktu linier. Kemudian, penerima pesan menentukan *Zq* dengan *q* yang merupakan bilangan prima dan sebuah pengali *r* elemen $Z_{q.}$ Kedua bilangan prima *q* dan *r* dapat dipilih secara acak dengan memenuhi syarat:

$$\boldsymbol{q > \sum_{i=1}^{n} w_i.} \qquad \boldsymbol{(2.4)}$$

Selanjutnya, penerima mentransformasikan vector *superincreasing knapsack w* menurut persamaan:

$$\boldsymbol{\beta_i \equiv w_i\, r (mod\ q).} \qquad \boldsymbol{(2.5)}$$

Deret himpunan tersebut akan menjadi kunci public dari sistem. Hal terpenting dalam algoritma Merkle-Hellman adalah nilai vector *w,* pengali *w,* dan bilangan prima *q* yang akan tetap dijaga rahasianya oleh penerima.

Ketika penerima menerima pesan kriptogram *c* atau pesan terenkripsi yang dibuat menurut persamaan (2.2), maka penerima pesan dapat mengkonversikan atau medeskripsikan kembali pesan tersebut dengan menemukan bilangan bulat **s** yang merupakan invers dari $\boldsymbol{r (mod\ q)}$ dimana $\boldsymbol{s\, r (mod\ q) = 1.}$ Untuk mencari nilai **s**, digunakan Algoritma *Extended Euclidean* dengan persamaan $= \boldsymbol{kq + 1}$. Selanjutnya penerima menghitung

$$\boldsymbol{c' \equiv c\, s \qquad (mod\ q)} \qquad \textbf{(2.6)}$$

Karenanya

$$c' \equiv cs \equiv \sum_{i=1}^{n} \alpha_i \beta_i s \qquad (mod\ q). \qquad (2.7)$$

Karena $s\,r(mod\ q) = 1$ mengikuti persamaan (2.5) maka

$$\beta_i.s \equiv w_i\, r\, s \equiv w_i \qquad (mod\ q) \qquad (2.8)$$

Karenanya

$$c' \equiv \sum_{i=1}^{n} a_i\, w_i \qquad (mod\ q) \qquad (2.9)$$

Jumlah semua nilai w_i lebih kecil dari q dan karenanya $\sum_{i=1}^{n} a_i w_i$ juga dalam interval [0, q-1]. Dengan demikian penerima harus menyelesaikan masalah jumlah himpunan bagian

$$c' = \sum_{i=1}^{n} a_i w_i \qquad (2.10)$$

Masalah ini mudah karena w adalah urutan *superincreasing*. Ambil elementerbesar dalam kata, misalnya $_k$. jika w $_k$ > c' maka a $_k$ = 0. Jika w $_k$ ≤ c' maka a $_k$ = 1. Kemudian, kurangi w $_k$ × a $_k$ dari c'. ulangi langkah ini hingga *chipertext* dapat di deskripsi.

2.3.4.1 Contoh Pembentukan Kunci

Perhitungan ini didasarkan pada algoritma *Merkle Hellman Knapsack.*Dengan cara membentuk kunci privat dan kunci publik terlebih dahulu. Berikut cara pembentukan kunci privat pada algoritma *Merkle Hellman Knapsack*:

1. Tentukan *w* atau barisan *superincreasing*. Misalnya { 2, 7, 11, 21, 42, 89, 180, 354}.
2. Tentukan nilai *q* yaitu 706 yang merupakan jumlah dari semua barisan *superincreasing*.

$$\sum w = 706$$

3. Tentukan nilai *r* misalnya 588 yang merupakan bilangan koprima dengan *q* dalam *range* 1 hingga *q*.

Untuk membentuk kunci publik pada algoritma *Merkle Hellman Knapsack,* hasilkan himpunan baru dengan mengalikan setiap elemen *superincreasing* dengan *r* mod *q*.Perhatikan contoh berikut:

Elemen $n = w_n \equiv (n \times r) \bmod q$

Elemen 1 = 2 $\equiv (2 \times 588) \bmod 881 = 295$

Elemen 2 = 7 $\equiv (7 \times 588) \bmod 881 = 592$

Elemen 3 = 11 $\equiv (11 \times 588) \bmod 881 = 301$

Elemen 4 = 21 $\equiv (21 \times 588) \bmod 881 = 14$

Elemen 5 = 42 $\equiv (42 \times 588) \bmod 881 = 28$

Elemen 6 = 89 $\equiv (89 \times 588) \bmod 881 = 353$

Elemen 7 = 180 $\equiv (180 \times 588) \bmod 881 = 120$

Elemen 8 = 354 $\equiv (354 \times 588) \bmod 881 = 236$

Himpunan {295,592,301,14,28,353,120,236} merupakan kunci publik yang akan digunakan untuk enkripsi.

Kunci publik akan dikirimkan kepada pengirim pesan untuk mengenkripsi pesan yang akan diterima oleh penerima pesan, sementara kunci privat tetap dijaga kerahasiaannya oleh penerima pesan guna mendekripsi pesan tersebut. Berikut langkah-langkah perhitungan untuk enkripsi dan dekripsi pesan tersebut.

2.3.4.2 Contoh Enkripsi

Tentukan pesan yang akan dienkripsi, misalnya huruf "a". Pertama terjemahkan "a" ke biner (dalam hal ini, menggunakan ASCII atau UTF-8) lalu mengalikan setiap bitnya masing dengan jumlah yang sesuai pada himpunan β.

a = 0 1 1 0 0 0 0 1

Kalikan tiap bit data biner dengan himpunan kunci publik yang telah dibentuk perhatikan tabel 4.1 berikut:

Tabel 2. 1 Perkalian Biner dan Kunci Publik

Bin	Kunci Publik	Hasil Perkalian
0	295	0
1	592	592
1	301	301
0	14	0
0	28	0
0	353	0
0	120	0
1	236	236
Jumlah		**1129**

Jumlah dari hasil perkalian tersebut merupakan *chipertext* yang akan dikirimkan kepada penerima pesan.

2.3.4.3 Contoh Dekripsi

Dekripsi merupakan cara untuk mengembalikan *chipertext* ke bentuk aslinya. Dekripsi dilakukan dengan menggunakan kunci privat yang terdiri dari:

w / *superincreasing* : {2,7,11,21,42,89,180,354}

q : 881

r : 588

Untuk mendekripsikan *chipertext* dilakukan beberapa langkah dekripsi dengan memanfaatkan kunci privat:

1. Tentukan nilai *s* mod *q* yang merupakan *inverst* dari *r* mod *q.* Cara mencari nilai *s* ialah dengan menggunakan algoritma *eucludian* dengan rumus $s = (1 + k \times q) / r$. dimana *s* haruslah bilangan bulat pertama dengan nilai $k = 1, 2, 3, \ldots,$ n. Dengan kunci privat yang ada, maka didapatkan nilai $k = 295$ dimana $s = (1 + 442 \times 881) / 588 = 442$.
2. Kalikan *chipertext* dengan *s* mod *q.* Dimana $1129 \times 442 \bmod 881 = 372$.
3. Bandingkan nilai 372 dengan nilai terbesar dalam himpunan *w.* jika *w* lebih besar maka tulis bit 0, jika *w* lebih atau sama kecil maka tulis bit 1 dan kurangi nilai tersebut dengan *w.* penulisan bit dimulai dari kiri ke kanan. Dalam contoh ini nilai *w* terbesar adalah 354. Dimana $372 > 354$ kemudian tulis bit 1 dan kurangi 372 dengan 354 sehingga nilai menjadi 18.
4. Ulangi langkah 3 hingga nilai telah dibandingkan dengan semua himpunan *w*. Contoh, $18 < 180$ maka tulis bit 0, bandingkan kembali, $18 < 89$ maka tulis bit 0, bandingkan kembali, $18 < 42$ maka tulis bit 0, bandingkan kembali, $18 < 21$ maka tulis bit 0, bandingkan kembali $18 > 11$ maka tulis bit 1 dan kurangi 18 dengan 11 sehingga nilai menjadi 7. Bandingkan kembali, $7 \leq 7$ maka tulis bit 1 dan kurangi 7 dengan 7 sehingga nilai 0. Bandingkan kembali, $0 < 2$ maka tulis bit 0. Setelah nilai dibandingkan dengan semua bobot *w* maka didapatkanlah kembali nilai bit sebelum dienkripsi yaitu : 0 1 1 0 0 0 0 1.

Ketika diterjemahkan kembali dari biner, pesannya akan kembali menjadi huruf “a”.

2.4 Algoritma *Huffman*

Algoritma *Huffman* dibuat oleh seorang mahasiswa MIT bernama David *Huffman* pada tahun 1952 yang merupakan salah satu metode lama dan yang paling terkenal dalam hal kompresi teks. Algoritma *Huffman* menggunakan prisip yang sama dengan kode morse, yaitu tiap karakter dikodekan hanya dengan rangkayan beberapa bit, dimana karakter yang sering muncul dikodekan dengan rangkayan bit yang pendek dan karakter yang jarang muncul dikodekan dengan rangkayan bit yang lebih panjang (Pahdi, 2017).

Algoritma *Huffman* termasuk dalam kelas algoritama yang menggunakan metode statik, hal ini dilihat dari tipe peta kode yang digunakan untuk merubah pesan awal menjadi kumpulan *codeword*. Metode statik selalu menggunakan peta kode yang sama, metode ini membutuhkan dua fase dalam algoritmanya: fase pertama, menghitung probabilitas kemunculan setiap simbol dan menentukan peta kodenya. Fase kedua, mengubah pesan menjadi kumpulan kode yang akan ditransmisikan.

Sedangkan teknik pengkodean untuk symbol, algoritma *Huffman* menggunakan metode *symbolwise*. Metode *symbolwise* merupakan metode yang menghitung peluang munculnya setiap symbol yang ada dalam satu waktu, diamana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan symbol yang jarang muncul.

2.4.1 Pembentukan Pohon *Huffman*

Pada dasarnya kode *Huffman* adalah kode prefiks (prefix code). Kode prefiks merupakan himpunan sekumpulan kode biner yang awal binernya tidak ada yang menjadi awal bagi kode biner yang lainnya. Kode prefiks biasanya digambarkan dengan pohon biner yang diberikan nilai atau label. Untuk cabang bagian kanan pohon biner diberi label 1, sedangkan untuk cabang bagian kiri diberi label 0. Rangkaian bit yang terbentuk di setiap lintasan dari akar ke daun pohon adalah kode prefiks karakter yang berpadanan. Pohon biner inilah yang disebut dengan pohon *Huffman*.

Berikut ini adalah langkah-langkah atau algoritma untuk membentuk pohon *Huffman*:

1. Hitunglah frekuensi munculnya setiap karakter dengan membaca semua indeks. Setiap karakter dinyatakan sebagai pohon yang bersimpul tunggal. Setiap simpul ditandai dengan frekuensi munculnya karakter tersebut.
2. Kemudian terapkan algoritma greedy dengan cara menggabungkan dua buah pohon yang memiliki frekuensi paling kecil pada sebuah akar. Setelah digabungkan, akar tersebut akan memiliki frekuensi yang merupakan jumlah frekuensi dua buah pohon penyusunnya.
3. Ulangi langkah 2 terus-menerus hingga hanya tersisa satu buah pohon *Huffman*. Semua pohon yang ada selalu terurut dan menaik berdasarkan frekuensi agar pemilihan dua pohon yang akan digabungkan berlangsung cepat.

Sebagai contoh, Kode ASCII *string* 8 huruf "MUHAMMAD" membutuhkan representasi 8 × 8 bit = 64 bit (8 byte), dengan rincian sebagai berikut:

M = 01001101

U = 01010101

H = 01000100

A = 01000001

M = 01001101

M = 01001101

A = 01000001

D = 01000100

Pada *string* di atas, frekuensi kemunculan M = 3, A =2, H = 1, U = 1 dan D = 1. Maka Pengerjaannya akan menjadi seperti berikut :

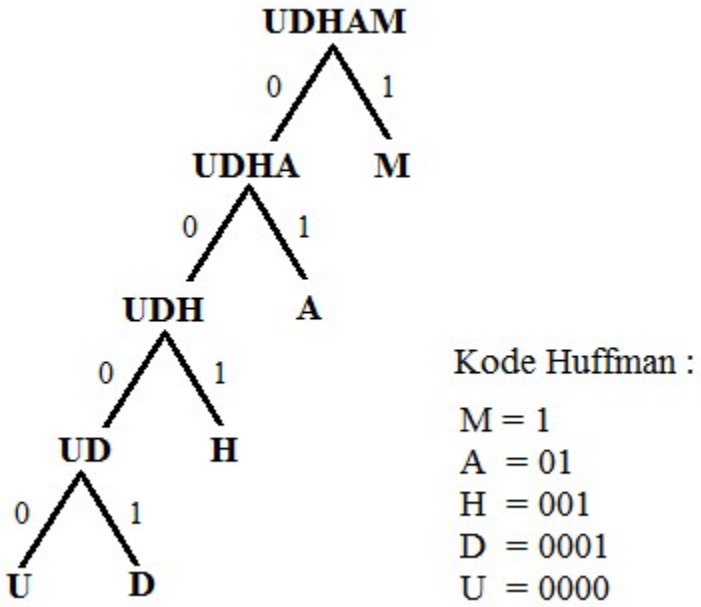
U dan D menjadi (UD) sehingga probabilitas menjadi 1/8 + 1/8 = 2/8

UD dan H menjadi (UDH) sehingga probabilitas menjadi 2/8 + 1/8 = 3/8

UDH dan A menjadi (UDHA) sehingga probabilitas menjadi 3/8 + 2/8 = 5/8

UDHA dan M menjadi (UDHAM) sehingga probabilitas saat ini menjadi 5/8 + 3/8 = 1.

Perlu diingat bahwa probabilitas lebih besar akan diletakkan di kiri dengan pengkodean 0, sehingga pohon *Huffman* dari soal tersebut adalah:



Gambar 2. 1 Pohon Huffman untuk Karakter "MUHAMMAD"

2.4.2 Proses Kompresi

Proses kompresi atau encoding adalah teknik penyusunan string biner dari teks yang ada. Sebelum proses encoding untuk karakter dilakukan, pohon *Huffman* harus dibuat terlebih dahulu. Setelah itu, buatlah kode untuk satu karakter dengan menyusun nama string biner yang terbaca dari akar hingga ke daun pohon *Huffman*.

Berikut ini adalah langkah-langkah atau algoritma untuk mengkompresi suatu string biner:

1. Tentukanlah karakter yang akan dikompresi.
2. Dimulai dari akar, bacalah setiap bit pada cabang yang saling sesuai hingga bertemu daun dimana karakter tersebut berada.
3. Ulangilah langkah 2 hingga seluruh karakter berhasil dikompresi.

Dengan melakukan pengkodean *Huffman* pada teks "MUHAMMAD" di atas kita dapat menghemat bit dari M yang sebelumnya 8 bit menjadi 1 bit, sehingga jumlah bit yang digunakan tadinya 8*5 bit = 40 bit menjadi 14 bit saja. Dapat dilihat jumlah bit setelah dilakukan pengkodean dengan algoritma *Huffman* dengan melihat Tabel 2.2 di bawah ini:

Tabel 2. 2 Kode Huffman untuk string "ACABBDA"

Huruf	Kode
M	1
A	01
H	001
D	0001
U	0000

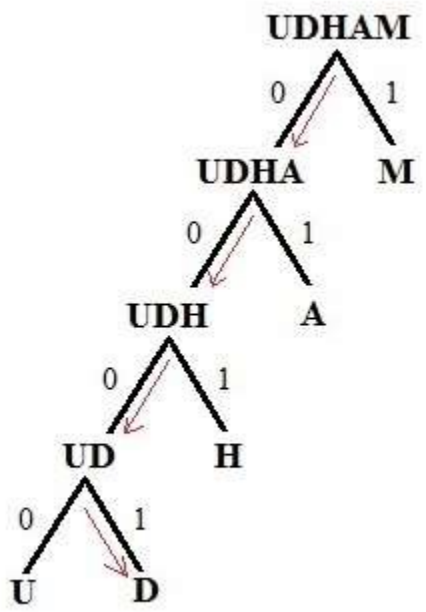
2.4.3 Proses Dekompresi

Proses dekompresi atau decoding merupakan proses untuk mengembalikan karakter yang telah dikompresi menjadi karakternya semula. Decoding berarti menyusun atau menata kembali data string biner menjadi sebuah karakter kembali. Dekompresi ini dapat dilakukan dengan dua cara, pertama dengan menggunakan pohon *Huffman*, kedua dengan menggunakan tabel kode *Huffman*.

Berikut adalah algoritma atau langkah-langkah untuk dekompresi string biner menggunakan pohon *Huffman*:

1. Baca sebuah bit dari string biner kemudian mulai dari akar.
2. Pada langkah 1 untuk setiap bitnya lintasilah pada cabang yang bersesuaian.
3. Ulangi langkah 1 dan 2 hingga bertemu daun, lalu kodekan rangkaian bit yang telah dibaca dengan karakter pada daun.
4. Ulangi terus-menerus dari langkah 1 hingga semua bit di dalam string habis.

Sebagai contoh kita akan men-decoding string biner yang bernilai “0001”:



Gambar 2. 2 Proses decoding string biner 0001.

Setelah kita telusuri dari akar, maka kita akan menemukan bahwa string yang mempunyai kode *Huffman* “0001” adalah karakter D.

Cara kedua adalah dengan menggunakan tabel kode *Huffman*. Perhatikan tabel berikut:

Tabel 2. 3 Contoh tabel kode Huffman

Karakter	String Biner *Huffman*
M	1
A	01
H	001
D	0001
U	0000

Dari Tabel 2.3 *string* tersebut akan direpresentasikan menjadi rangkaian bit:

1 0000 001 01 1 1 01 0001.

Jadi, jumlah bit yang dibutuhkan hanya 18 bit.

Dari Tabel 1 tampak bahwa kode untuk sebuah simbol/karakter tidak boleh menjadi awalan dari kode simbol yang lain guna menghindari keraguan (ambiguitas) dalam proses dekompresi atau *decoding*. Karena tiap kode *Huffman* yang dihasilkan unik, maka proses *decoding* dapat dilakukan dengan mudah.

Contoh:

Saat membaca kode bit pertama dalam rangkaian “1000000101110100 01”, yaitu bit “1”, dapat langsung disimpulkan bahwa kode bit “1” merupakan pemetaan dari simbol “M”. Kemudian baca kode bit selanjutnya, yaitu bit “0”. Tidak ada kode *Huffman* “0”, lalu baca kode bit selanjutnya, sehingga menjadi “00”. Tidak ada juga kode *Huffman* “00”, lalu baca lagi kode bit berikutnya, sehingga menjadi “000”, lalu baca lagi kode bit berikutnya, sehingga menjadi “0000”. Rangkaian kode bit “0000” adalah pemetaan dari symbol “U” sehingga karakter kedua adalah “U”. Berikutnya adalah “001” ditemukan bahwa karkter berikutnya adalah “H”. Berikutnya “01” sehingga diketahui karakter “A”.

Kemudian “1” berulang 2 kali menandakan “MM”. Kemudian “01” lagi yaitu karakter “A”. Hingga “0001” untuk kode “D” yang terakhir, sehingga diperoleh kembali bahwa string input adalah “MUHAMMAD”.

2.5 Matlab

Matlab atau *Matrix Laboratory* adalah sebuah program atau aplikasi komputasi numerikal dan bahasa pemograman komputer generasi keempat yang dikembangkan oleh MathWork. Diawal perkembangannya Matlab berkerja dengan data berupa matrik yang berasal dari excel atau file berekstensi dat yang diketik dengan teks editor seperti Notepad, wordpad, dan sejenisnya (Handayanto & Herlawati, 2018).

Matlab dapat memanipulasi matrik, membuat plot fungsi dan data, implementasi algoritma, dan membuat antar muka pengguna. Dalam penelitian ini algoritma *Merkle Helman Knapsack* akan di implementasikan menggunakan Matlab.

2.6 Penelitian Terkait

Berikut ini adalah penelitian terkait yang pernah dilakukan sebelumnya tentang pengolahan file suara, algoritma Merkle-Hellman Knapsack serta algoritma *Huffman* pada kompresi teks:

1. Pemetaan dan Analisis Tipe Suara Manusia Menggunakan Fast Fourier Transform ditulis oleh Rini Mulyani pada tahun 2017 dengan kesimpulan bahwa *range* frekuensi yang dihasilkan pada setiap relawan bergantung pada *vocal range* yang diperoleh, semakin besar *vocal range* yang diperoleh maka semakin besar *range* frekeunsi yang dihasilkan (Mulyani, 2017).
2. Identifikasi Suara dengan MATLAB sebagai Aplikasi Jaringan Syaraf Tiruan ditulis oleh Jhon Alder, Muhamad Azhar, dan Sri Supatmi pada tahun 2013 dengan kesimpulan Jaringan Syaraf Tiruan mampu memvisualisasikan suatu proses sinyal suara menjadi sinyal frekuensi (Adler, Azhar, & Supatmi, 2013).

3. Analisa Kombinasi Algoritma Merkle-Hellman Knapsack dan Logaritma Diskrit pada Aplikasi Chat yang ditulis oleh Aminudin, Ahmad Faisal Helmi, dan Sofyan Afrianto dengan kesimpulan penelitian bahwa algortima knapsack memiliki performa yang lebih cepat dibandingkan kombinasi knapsack dan logaritma diskrit. Hal tersebut dikarenakan pada kombinasi knapsack proses pembangkitan kunci, enkripsi dan dekripsi dilakukan dua kali proses sehingga waktu eksekusi juga ikut berpengaruh (Aminudin, Helmi, & Afrianto, 2018).
4. *Encrypting Messages using the Merkle-Hellman Knapsack Cryptosystem* yang ditulis oleh Ashish Agarwal pada tahun 2014 dengan kesimpulan penelitian dengan algoritma Merkle Hellman Knapsack pesan teks "Hello" berhasil dienkripsi dan dideskripsi kembali menjadi pesan semula dengan baik (Agarwal, 2011).
5. Perancangan Aplikasi Keamanan Data Teks Menggunakan Algoritma Merkle-Hellman Knapsack yang ditulis oleh Murdani pada tahun 2017 dengan kesimpulan penelitian bahwa enkripsi dan dekripsi menghasilkan *chipertext* berupa deretan angka dan menghasilkan *plaintext input* yang sama dengan *plaintext output* dari proses dekripsi. Enkripsi ini lebih aman dibandingkan metode kriptografi yang menghasilkan enkripsi dalam bentuk teks (Murdani, 2017).
6. *Enhancement of Merkle-Hellman Knapsack Cryptosystem by Use of Discrete Logarithmics* yang ditulis oleh Arghya Ray dan Santhoshi Bhat pada tahun 2013 dengan kesimpulan penelitian bahwa percobaan enkripsi dengan kata "get" kemudian didekripsikan menghasil kata yang sama dengan kata sebelum dienkripsi (Ray & Bhat, 2013).
7. Perbandingan Tiga Langkah Teknik-Teknik Kompresi Teks yang ditulis oleh Adam Puspabhuana pada tahun 2016 dengan kesimpulan penelitian bahwa kombinasi dua algoritma dalam dua langkah seperti bit *reduction* dan *Huffman* meningkatkan rasio kompresi secara signifikan (Puspabhuana, 2016).

8. Implementasi Algoritma Kompresi Data Huffman untuk Memperkecil Ukuran File MP3 Player yang ditulis oleh Victor Amrizal pada tahun 2010 dengan kesimpulan pengujian bahwa algoritma *Huffman* mempunyai rasio kompresi yang cukup tinggi (Amrizal, 2010).
9. Penerapan Algoritma Huffman dalam Dunia Kriptografi yang ditulis oleh Yogie Adrisatria pada tahun 2015 dengan kesimpulan penelitian bahwa penerapan algoritma *Huffman* dalam kriptografi cukup layak bila menggunakan alternatif penerapan yang ada karena pertimbangan dari segi keamanan yang cukup mumpuni dan dari segi kunci yang cukup praktis (Adrisatria, 2015).
10. Implementasi Teknik Kompresi Teks *Huffman* yang ditulis oleh Andysah Putera Utama Siahaan pada tahun 2016 dengan kesimpulan penelitian bahwa *Huffman encoding* sangat ampuh untuk pesan teks yang memiliki karakter yang sama (Siahaan, 2016).
11. Algoritma *Huffman* dan Kompresi Data yang ditulis oleh David Soendoro pada tahun 2013 dengan kesimpulan bahwa algoritma *Huffman* terbukti ampuh dalam melakukan kompresi data, walaupun algoritma *Huffman* sudah sangat lama diciptakan (Soendoro, 2013).
12. Implementasi Algoritma *Huffman* dan LZ78 untuk Kompresi Data ditulis oleh Gusri Indah Yana dan Rivalri Kristianto Hondro pada tahun 2016 dengan kesimpulan hasil penelitian bahwa dalam hal kompresi teks algoritma *Huffman* lebih baik dari pada algoritma LZ78 (Yana & Hondro, 2016).
13. Impelementasi Kompresi Teks Menggunakan Metode *Huffman* untuk Menghemat Karakter pada Short Message Service ditulis oleh Evi Mariani Harahap, Dian Rachmawati, S.Si, M.Kom, dan Herriyance, ST, M.Kom dengan kesimpulan kompresi serta dekompresi pada teks SMS menggunakan metode Huffman dapat menghemat pesan SMS yang akan dikirimkan, rasio pesan teks yang ditulis dapat menghasilkan persentase hingga 31% dan terhadap contoh default yang dibuat (Harahap, Rachmawati, & Herriyance, 2013).